DOI 10.31891/2307-5732-2021-297-3-25-29
УДК 004.053:004.054:004.052.2

О. В. БАРМАК, В. В. КУДРЯВЦЕВ, Ю. В. ФОРКУН, О. М. ЯШИНА
Хмельницький національний університет

ПІДХІД ДО АНАЛІЗУ ПРОГРАМНОГО КОДУ З ВИКОРИСТАННЯМ МЕТРИК ХОЛСТЕДА

У роботі наведено результати досліджень різних стандартів, правил та методик написання програмного коду та аналізу їх впливу на якість ПЗ й імовірність виникнення технічних ризиків, пов'язаних з інформаційними процесами всередині системи.

Ключові слова: розробка програмного забезпечення, оцінка ризиків, стандарти в програмуванні, метрики коду, забезпечення якості.

A. V. BARMAK, V. KUDRIAVTSEV, Y. V. FORKUN, O. M. YASHYNA
Khmelnytskyi National University

SOFTWARE CODE ANALYSIS SYSTEM FOR RISK ASSESSMENT AND QUALITY ASSURANCE OF SOFTWARE

The paper presents the results of research of various standards, rules and methods of writing software code and analysis of their impact on software quality and the likelihood of technical risks associated with information processes within the system. Most of the risks that arise while developing software products are due to errors in building the system architecture or writing code. As a solution for such problems, it is proposed to apply the developed set of rules and methods to build the system architecture and assess the quality of writing software objects. Metrics have been developed to estimate the size and complexity of the module by combining elements of Halsted and Chepin metrics. Also, a set of principles for optimizing the structure of the system, also known as SOLID principles, was presented. The application of these principles for system construction and analysis was substantiated in order to minimize risks, ensure the quality of the software system and provide opportunities for easy extensibility of the project. Using these methods will optimize the project both for use and for further development. The need for such optimization processes in terms of risk management is that the clearer the system and the easier it is to expand, the less likely it is that errors will occur in the future when adding new functionality.

Keywords: Software development, risk assessment, standards in programming, code metrics, quality assurance.

Постановка проблеми

На сьогодні у більшості ІТ-проектів все більше приділяється увага якісному написанню коду. Поняття якісного коду у сучасному його розумінні означає не лише програмний код, який виконує поставлену задачу, але і відповідає певним стандартам, а також є легким для розуміння.

Причиною такої еволюції поняття є, в першу чергу, розвиток проблем, які ставляться перед ІТ-галуззю, а відповідно і збільшення об'ємів ІТ-проектів. Також великі проекти вимагають залучення більшої кількості людей, що призводить до зростання важливості збереження єдиного стилю написання коду.

Як наслідок, було розроблено багато різноманітних стандартів програмування, які надають список правил та рекомендацій стосовно написання коду та побудови архітектури програмних систем. Сучасні автори [1] наводять такі переваги стандартизації в програмуванні, як збільшення ефективності, полегшення підтримки та зменшення складності коду. В той же час, відхилення від загальноприйнятих стандартів програмування може вилитись у серйозні проблеми з продуктивністю та захищеністю програмного забезпечення. Впровадження стандартів програмування може не дати позитивних результатів одразу. При впровадженні нового стандарту в проект розробникам може знадобитись певний час, щоб звикнути до нього та адаптувати свій стиль коду до нових правил. Проте, авторами роботи [2] було встановлено що запровадження стандартів з часом зменшує імовірність виникнення помилки у проекті, а відповідно покращує ефективність роботи системи.

Сучасними авторами також відзначається висока ефективність використання метрик в області прогнозування якості програмних продуктів [9]. Метрики поділяють на метрики процесів та метрики кодування [11]. Метрики процесів відносяться до властивостей самого процесу розробки, а метрики кодування – безпосередньо до програмного продукту, який розробляється. Метрики кодування, такі як метрики Холстеда, на сьогоднішній день часто використовуються для аналізу якості програмних систем. Такі метрики застосовуються, зокрема і для розробки аналізаторів коду, які оцінюють якість інших програм [8]. До переваг подібних метрик відносять їх низьку залежність від мов програмування – вони оперують найбільш базовими елементами, які наявні майже в усіх сучасних мовах [12]. Отже, існує простір для покращення ефективності впровадження стандартів програмування, особливо на початкових етапах. Одним з можливих рішень для такої оптимізації є розробка системи методів та правил для аналізу програмного коду. Застосування такої системи дозволить оцінювати якість складових елементів проекту, що забезпечить можливість прийняття рішення про оптимізацію програмного коду та процесів розробки.

Метою роботи є розробка системи методів та правил для аналізу програмного коду з метою забезпечення якості програмного забезпечення (ПЗ) та оцінки ризиків проекту.

Результати досліджень

В результаті проведеного аналізу можна стверджувати, що на сьогоднішній день існує багато різноманітних стандартів та принципів програмування, які дозволяють створити певну систему правил написання коду та побудови архітектури системи, відповідно до потреб проекту.

Аналіз програмного коду для оцінки ризиків і забезпечення якості програмного забезпечення слід починати з проектування програмних систем з дотриманням принципів SOLID [4]. Об'єктно-орієнтоване програмування вводить в розробку ПЗ ряд специфічних підходів до проектування програмних систем. Однією з переваг об'єктно-орієнтованого програмування (ООП) є те, що розробник має змогу виділити сутності та алгоритми, призначені для розв'язання однієї задачі в окремі класи. Проте, саме по собі застосування об'єктно-орієнтованої парадигми розробки не означає, що розробник застрахований від написання заплутаного і незрозумілого коду. Для вирішення такої проблеми Робертом Мартіном було сформульовано п'ять принципів об'єктно-орієнтованого програмування та проектування, які на сьогодні відомі під акронімом SOLID, який можна розшифрувати наступним чином:

– Single responsibility: принцип єдиної відповідальності;
– Open-closed: принцип відкритої закритості;
– Liskov substitution: принцип підстановки Барбари-Лісков;
– Interface segregation: принцип розділення інтерфейсів;
– Dependency inversion: принцип інверсії залежності.

Принцип єдиної відповідальності в системі аналізу програмного коду полягає в тому, що клас повинен бути призначений для чогось одного. Збереження цього принципу призведе до поділу системи на модулі з чітко визначеним функціоналом. Наявність чітко розмежованої системи зробить архітектуру проекту більш зрозумілою та легшою до розширення.

Принцип відкритої закритості в системі аналізу також дозволить зменшити імовірність помилок. Розширення функціоналу класу, замість його кардинальної зміни забезпечує збереження правильної роботи вже існуючого функціоналу.

Принцип підстановки Лісков теж відіграє важливу роль в системі забезпечення якісного коду. Цей підхід також призначений для безпечного розширення функціоналу системи. Перевагою дотримання цього принципу є збереження ключової функціональності в сімействах класів, що покращує розуміння системи в цілому.

Впровадження принципу розділення інтерфейсів в систему аналізу коду дозволить підтримувати ієрархію класів програмної системи у логічному і оптимізованому вигляді. Порушення цього принципу буде сприяти перевантаженню класів непотрібним, або некорисним функціоналом, що буде негативно відбиватися на зрозумілості коду та може сприяти виникненню непередбачуваних помилок.

Принцип інверсії залежностей також відіграватиме значну роль в системі. Впровадження такого принципу має на меті підвищення модульності програмної системи. Програмний продукт, який вдало розділений на модулі значно легше підтримувати та модифікувати ніж той, який представляє собою велику монолітну систему.

Хоча визначення принципів SOLID оперують багатьма термінами з об'єктно-орієнтованої парадигми програмування, вони також можуть бути застосовані і для інших парадигм. Наприклад, багато експертів в області програмування погоджуються, що ці принципи можуть бути застосовані і в області функційного програмування. Також варто зазначити, що принцип єдиної відповідальності був відомий ще за часів імперативної парадигми програмування. Тому, можна стверджувати, що принципи SOLID можна застосовувати не тільки при використанні об'єктно-орієнтованої парадигми, якщо того потребує проект, хоча і основною областю їх застосування залишаються системи з об'єктно-орієнтованою архітектурою.

На сьогоднішній день SOLID є одним із найбільш поширених принципів для об'єктно-орієнтованого програмування. Хоча застосування цих принципів може збільшити об'єм коду та складність розробки, проте слідування SOLID значно покращує якість коду та спрощує підтримку системи. Дослідження показали, що застосування цих принципів може зменшити зчеплення у проекті на 69 % і збільшити зв'язність до 29 % [3].

Таким чином, першим кроком алгоритму використання системи аналізу ризиків і забезпечення якості програмного коду буде застосування принципів SOLID при розробці програмної системи. Це забезпечить якість та зрозумілість програмного коду та поділить проект на легко розширювані модулі.

Кількість отриманих модулів залежить від масштабів розроблюваного проекту. Найпростіші проекти можуть складатись з невеликої кількості модулів. Проекти більшого масштабу можуть складатись з великої кількості модулів. В деяких випадках проект може складатись з декількох підпроектів різної величини – такий підхід називається мікросервісною архітектурою [13].

Після розроблення певної кількості модулів проекту, виникає потреба в оцінці їх якості. Для цих цілей було розроблено багато метрик програмного забезпечення. Метрика програмного забезпечення – це міра, що дозволяє отримати числове значення певних властивостей програмного модуля. В цілому, метрики призначені для визначення складності проекту та прогнозування об'ємів робіт.

Найпростішим класом метрик ПЗ є кількісні метрики. Ці метрики призначені для визначення кількісних характеристик вихідного коду. Класичним представником цього класу є метрика кількості рядків коду (SLOC – Source lines of code). Ця метрика може використовуватись для визначення трудозатрат по проекту. Проте, з появою мов програмування що підтримують запис декількох команд в одному рядку, ця метрика стала застарілою. Їй на зміну прийшла метрика кількості логічних рядків коду, тобто кількість команд. Ще однією можливою альтернативою вимірювання цієї метрики може бути підрахунок кількості операторів в коді. Таким чином, визначається не тільки об'єм коду, але і в певній мірі і об'єм виконуваної програмою роботи. Проте, усі ці метрики значно залежать від особливостей мови програмування, оскільки

різні мови можуть потребувати різної кількості рядків та операторів. Тому, при оцінці цих метрик важливо враховувати специфіку використаної мови. В зв'язку з цим, ці метрики погано підходять для порівняння програм, написаних різними мовами.

Також, до кількісних метрик відносять метрики Холстеда [5]. При використанні метрик Холстеда частково компенсуються недоліки, пов'язані з записом однієї і тієї ж операції різною кількістю рядків. Багато авторів відзначають метрику Холстеда як один з найпоширеніших методів прогнозування надійності програм [9]. Метрику Холстеда можна вдосконалити, скориставшись методами з метрики Чепіна [6]. Обчислення метрики Чепіна передбачає врахування вагових коефіцієнтів, які залежать від типів змінних. Метрика Чепіна поділяє змінні на 4 типи – введені змінні для розрахунків та забезпечення виводу; модифіковані, або створені всередині програми змінні; управляючі змінні; невикористані змінні. Вагові коефіцієнти, на думку автора метрики, розподіляються наступним чином – $a_1 = 1$, $a_2 = 2$ $a_3 = 3$, $a_4 = 0.5$. Ці самі коефіцієнти можна враховувати при обчисленні кількості операндів в метриці Холстеда. Аналогічно, для підрахунку операторів можна також використати вагові коефіцієнти, які будуть рівні кількості аргументів оператора. Таким чином, формули метрики Холстеда приймуть наступний вигляд:

$$n_1 = a_1 p + a_2 m + a_3 c + a_4 t \quad (1)$$

$$n_2 = \sum k_i i \quad (2)$$

$$N_1 = a_1 P + a_2 M + a_3 C + a_4 T \quad (3)$$

$$N_2 = \sum K_i i \quad (4)$$

де p – кількість змінних для розрахунків та забезпечення виводу, m – кількість модифікованих, або створені всередині програми змінних, c – кількість управляючих змінних, t – кількість невикористаних змінних, k_i – кількість операторів з кількістю аргументів i, i – кількість аргументів, p – кількість використань змінних для розрахунків та забезпечення виводу, m – кількість використань модифікованих, або створені всередині програми змінних, c – кількість використань управляючих змінних, t – кількість повторів невикористаних змінних, k_i – кількість використань операторів з i аргументів.

Таким чином, метрика Холстеда буде враховувати не тільки кількісні характеристики коду, але і складність потоку управління даними. Такий підхід дозволить більш точно оцінити складність класу або модуля програмної системи. Варто зазначити, що єдиної універсальної метрики не існує. Будь-які метричні характеристики програми повинні контролюватись в залежності від поставленої задачі та інших метрик. Будь-яка метрика – це лише показник, який в певній мірі може служити індикатором якості програмного коду, проте не варто зводити одну метрику в абсолют і приймати рішення опираючись лише на неї.

Таким чином, можна побудувати систему принципів написання та оцінки програмного коду для забезпечення якості та розширюваності проекту. В якості вихідної точки прийматиметься об'єктно-орієнтована парадигма, оскільки на сьогоднішній день вона найбільш розповсюджена в програмних проектах різних масштабів. За основу слід взяти набір принципів SOLID. Слідування цим принципам забезпечить зрозумілість проекту як на рівні архітектури, так і на рівні окремих модулів та класів. У якості певного мірила складності системи варто взяти метрики коду, наприклад розширену версію метрики Холстеда, яка була представлена вище. Метрики не варто розглядати в контексті лише одного, окремо взятого модуля або класу [7]. Коректним застосуванням значень метрик буде порівняння їх з метриками інших класів та модулів системи. Наприклад, якщо метрики якогось модуля значно більше ніж в середньому по проекту, то це може свідчити про потребу в реорганізації алгоритмів в цьому модулі, або проведенні загального рефакторингу, якщо цей модуль тісно пов'язаний з іншим функціоналом.

Роботу розробленої системи можна зобразити за допомогою схеми, наведеної на рис. 1.

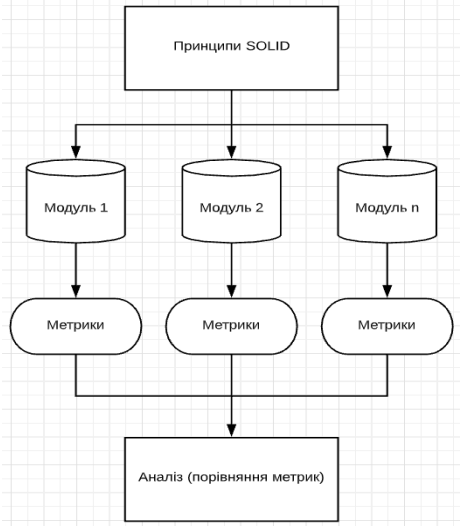

Рис. 1. Схема роботи системи

Розробка програмних модулів проекту ведеться з дотриманням принципів SOLID. Далі, для кожного модуля вираховуються метрики, і на основі порівняння цих метрик проводиться аналіз та приймається рішення про якість модулів та потребу в рефакторингу. Варто зазначити, що ця схема роботи має рекурсивний характер, тобто вона можу бути застосована як на рівні модулів системи, так і на більш низькому рівні всередині самих модулів, по відношенню до підсистем та класів, які належать цьому модулю.

Загалом, така система принципів може бути розширена, відповідно до потреб проекту. Наприклад, можуть бути додані інші метрики для більш детального контролю якості. Але, в загальному, наведеної сукупності принципів і метрик повинно бути достатньо для підтримання якості проектів різних масштабів.

Висновки

Таким чином, в роботі було проведено аналіз сучасних стандартів та метрик програмування. В тому числі, було запропоновано варіант вдосконалення метрики Холстеда, що дозволяє метриці враховувати не тільки кількісні параметри коду, але і складність потоку даних. Також, розроблено систему аналізу програмного коду для забезпечення якості програмних продуктів та оцінки ризиків.

Представлена система фактично базується на використанні відомих принципів та методів і основною її перевагою є те, що вона не тільки наводить основні вказівки з написання програмного коду, але і дозволяє динамічно оцінювати структуру модулів розроблюваної програмної системи, що дозволяє вчасно приймати рішення про необхідність проведення оптимізаційних процесів та рефакторингів над модулями програмної системи. Отже, застосування такої системи аналізу є досить доцільним для проектів з великою кількістю модулів з метою збереження якості програмного продукту та мінімізації ризиків, пов'язаних з помилками в коді або архітектурі системи.

Література

1. Importance of Code Quality and Coding Standard in Software Development. multidots.com. 2020. URL: https://www.multidots.com/importance-of-code-quality-and-coding-standard-in-software-development/.

2. Srđan Popić, Gordana Velikic, Hlavač Jaroslav, Zvjezdan Spasic Pavkovic. The Benefits of the Coding Standards Enforcement and its Impact on the Developers Coding Behaviour-A Case Study on Two Small Projects : тези конф. (м. Белград, листопад 2018 р.). Белград, 2018. – URL : https://www.researchgate.net/publication/328912784_The_Benefits_of_the_Coding_Standards_Enforcement_and_its_Impact_on_the_Developers_Coding_Behaviour-A_Case_Study_on_Two_Small_Projects.

3. Effect of SOLID Design Principles on Quality of Software: An Empirical Assessment. International Journal of Scientific & Engineering Research. 2015.

4. Martin R. Design Principles and Design Patterns. 2000. 32 p.

5. Halstead, Maurice H. Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7 1977.

6. Chapin N. An entropy metric for software maintainability. In System Sciences. Vol. II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on. Vol. 2, pp. 522–523.

7. Fedasyuk D., Yakovyna V., Serdyuk P., Nytrebych O. Variables state-based software usage model. ECONTECHMOD. AN INTERNATIONAL QUARTERLY JOURNAL. 2014. Vol. 3. No. 2. P. 15–20.

8. Афанасова А.И. Программа по оценке качества академических программных продуктов на основе методики Холстеда / А.И. Афанасова // Программные продукты и системы. – 2015. – № 4 (112). – С. 256–260.

9. Аверьянов А. В. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ / А. В. Аверьянов, И. Н. Кошель, В. В. Кузнецов // Известия высших учебных заведений «Приборостроение». – 2019. – № 11.

10. Цветков В.Я. Метрики сложной детерминированной системы / В.Я. Цветков, А.В. Буравцев // Онтология проектирования. – 2017. – Т. 7, № 3(25). – С. 334–346.

11. Singh G., Singh D., Singh V. A Study of Software Metrics. IJCEM International Journal of Computational Engineering & Management. 2011. Vol. 11. P. 22–27.

12. Thirumalai C., Thirunavukkarasu H., Vidhyagaran G., Seenu K. Software Complexity Analysis Using Halstead Metrics. International Conference on Trends in Electronics and Informatics – ICEI 2017.

13. Клапчук Р. Г. Монолітні веб-сервіси та мікросервіси: порівняння та вибір / Р. Г. Клапчук, В. С. Харченко // Радіоелектронні і комп'ютерні системи. – 2017. – № 1 (81).

References

1. Importance of Code Quality and Coding Standard in Software Development. multidots.com. 2020. URL: https://www./multidots.com/importance-of-code-quality-and-coding-standard-in-software-development/.

2. Srđan Popić, Gordana Velikic, Hlavač Jaroslav, Zvjezdan Spasic Pavkovic. The Benefits of the Coding Standards Enforcement and its Impact on the Developers Coding Behaviour-A Case Study on Two Small Projects : tezy konf. (m. Belhrad, lystopad 2018 r.). Belhrad, 2018. – URL : https://www.researchgate.net/publication/328912784_The_Benefits_of_the_Coding_Standards_Enforcement_and_its_Impact_on_the_Developers_Coding_Behaviour-A_Case_Study_on_Two_Small_Projects.

3. Effect of SOLID Design Principles on Quality of Software: An Empirical Assessment. International Journal of Scientific & Engineering Research. 2015.

4. Martin R. Design Principles and Design Patterns. 2000. 32 p.

5. Halstead, Maurice H. Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7 1977.

6. Chapin N. An entropy metric for software maintainability. In System Sciences. Vol. II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on. Vol. 2, pp. 522–523.

7. Fedasyuk D., Yakovyna V., Serdyuk P., Nytrebych O. Variables state-based software usage model. ECONTECHMOD. AN INTERNATIONAL QUARTERLY JOURNAL. 2014. Vol. 3. No. 2. P. 15–20.

8. Afanasova A.I. Programma po ocenke kachestva akademicheskih programmnyh produktov na osnove metodiki Holsteda / A.I. Afanasova // Programmnye produkty i sistemy. – 2015. – № 4 (112). – S. 256–260.

9. Averyanov A. V. Primenenie metrik Holsteda dlya kolichestvennogo ocenivaniya harakteristik programm EVM / A. V. Averyanov, I. N. Koshel, V. V. Kuznecov // Izvestiya vysshih uchebnyh zavedenij «Priborostroenie». – 2019. – № 11.

10. Cvetkov V.Ya. Metriki slozhnoj determinirovannoj sistemy / V.Ya. Cvetkov, A.V. Buravcev // Ontologiya proektirovaniya. – 2017. – T. 7, № 3(25). – S. 334–346.

11. Singh G., Singh D., Singh V. A Study of Software Metrics. IJCEM International Journal of Computational Engineering & Management. 2011. Vol. 11. P. 22–27.

12. Thirumalai S., Thirunavukkarasu H., Vidhyagaran G., Seenu K. Software Complexity Analysis Using Halstead Metrics. International Conference on Trends in Electronics and Informatics - ICEI 2017.

13. Klapchuk R. H. Monolitni veb-servisy ta mikroservisy: porivniannia ta vybir / R. H. Klapchuk, V. S. Kharchenko // Radioelektronni i kompiuterni systemy. – 2017. – № 1 (81).

О. В. БАРМАК ORCID ID: 0000-0003-0739-9678 alexander.barmak@gmail.com
В. В. КУДРЯВЦЕВ vivern07@gmail.com
Ю. В. ФОРКУН ORCID ID: 0000-0002-7906-4191 forkun@ridne.net
О. М. ЯШИНА ORCID ID: 0000-0001-7816-1662 ksusha.ja@gmail.com

Рецензія/Peer review : 18.04.2021 р. Надрукована/Printed :30.06.2021 р.